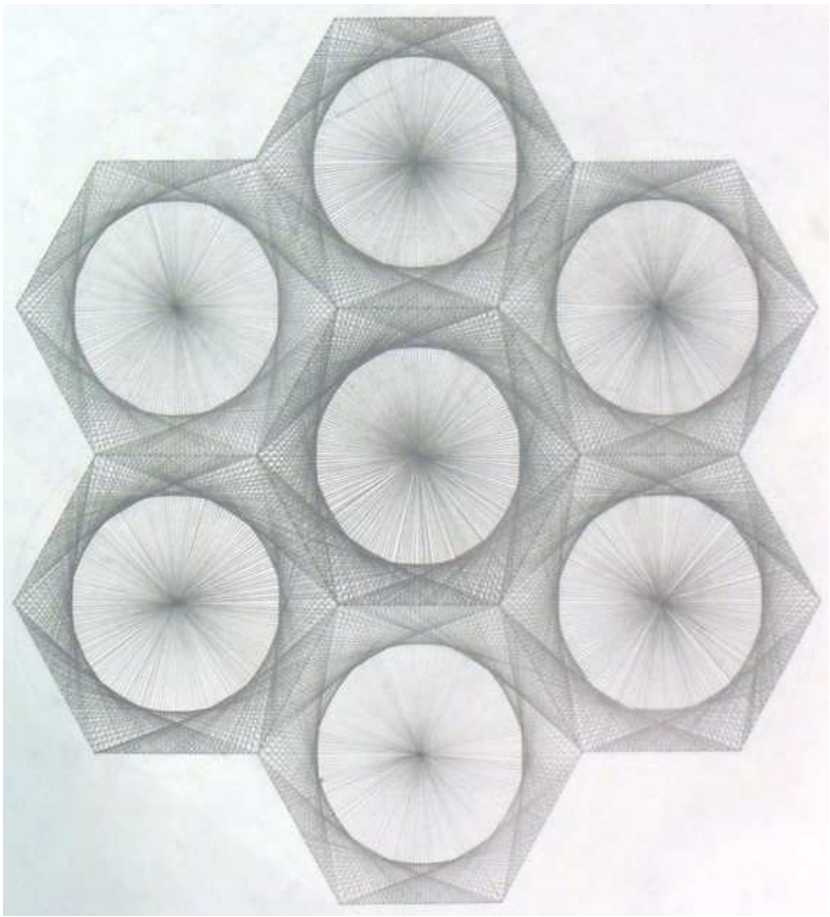
PRÁCTICA 6. RECURSIVIDAD

1 sesión

Semana: 4 de Noviembre

OBJETIVOS:

- ✓ Comprender el concepto de módulo recursivo
- ✓ Comprender la ejecución de un módulo recursivo mediante la realización de trazas
- ✓ Utilizar métodos recursivos para la resolución de problemas que pueden definirse de modo natural en términos recursivos, tales como muchas funciones matemáticas



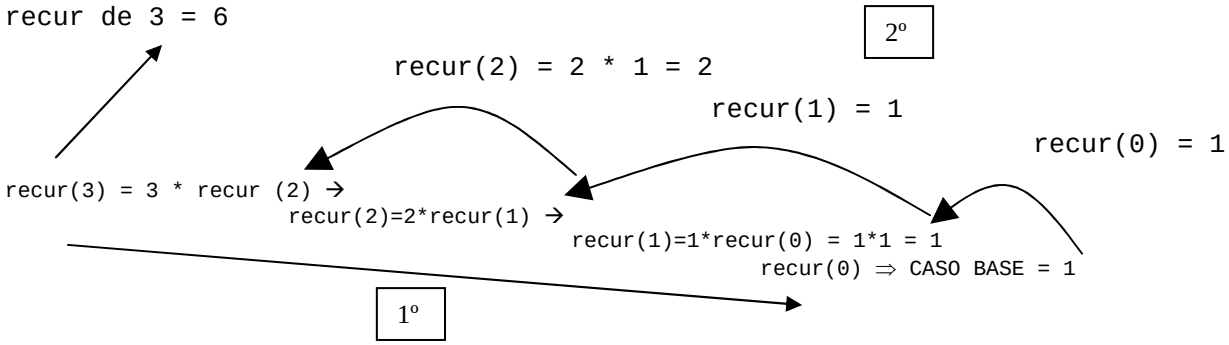
Ejercicio Resuelto 1. **Realiza varias trazas de la siguiente función recursiva y describe en lenguaje natural qué calcula la función**.

```
int  recur(int n)
{
    int result;
    if (n==0)
         result = 1;
    else
         result = n * recur(n – 1);

    return(result);
}
```

En esta traza :

1. Se producen llamadas para calcular ***recur(n)*** con n decreciendo sucesivamente. La resolución de ***recur(n)*** se queda en espera, hasta que se calcule ***recur(n - 1)***
2. Se llega a un caso base, tras el cual se produce una vuelta a atrás en la secuencia anterior, para terminar la resolución de las distintas llamadas que se habían quedado en espera.



Esta función recursiva calcula el factorial de un número.

Ejercicio Resuelto 2. **Realiza un programa en C que pida por teclado n números enteros y los muestre por pantalla (recursivamente) en sentido contrario al leído.**

Ejemplo:

```
Introduce número: 34
Introduce número: 7
Introduce número: 102
Introduce número: 65
65
102
7
34
```

El procedimiento recursivo se puede implementar de la siguiente forma:

```
void invertir(int n)  // n es el total de números a introducir
{
  int num;

  if(n>0)
```

```
    {
            cout << "Introduce número: ";
            cin >> num;

            invertir(n-1);
            cout << num << endl;
    }
}
```

¿Qué sucedería si hacemos la llamada recursiva después de imprimir el número por pantalla?

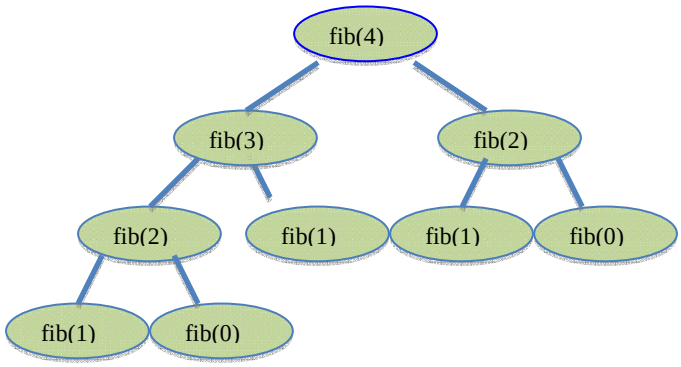
Ejercicio Resuelto 3. **Implementa un módulo recursivo en C que, dado un número en decimal, imprima por pantalla su correspondiente binario.**

Está basado en el método que usamos para convertir un número decimal a binario "a mano": dividir el número entre dos hasta el caso base e imprimir los restos del último al primero:

```
void decimalABinario(int n)
{
    if(n<=1)
         cout << n;
    else
    {
         decimalABinario(n/2);
         cout << n%2;
    }
}
```

Ejercicio Resuelto 4. **Implementa en C un programa recursivo que calcule el valor de la serie de Fibonacci para un número n dado. Recuerda que la serie de Fibbonacci es la siguiente: 1 1 2 3 5 8 13…**

Árbol de llamadas recursivas para fib(4):



Su definición matemática es:

$$fib(n) = \begin{cases} fib(n) = 1, \text{ si } n=0 \text{ ó } n=1 \\ fib(n) = fib(n-2) + fib(n-1) \end{cases}$$

La función recursiva que calcula esta fórmula se implementa de la siguiente forma:

```
int fib(int n)
{
  int resultado;

  if(n == 0 || n == 1)
           resultado = 1;
  else
           resultado = fib(n-1) + fib(n-2);

  return resultado;
}
```

Ejercicio Resuelto 5. **Dada la siguiente función en C que devuelve la suma de los dígitos del número pasado como argumento, implementa una función equivalente recursiva:**

```
int sumaDigitos(int n)
{
        int res = 0;

        while(n>9)
        {
              res += n%10;
              n = n / 10;
        }
        res += n;
        return res;
}
```

Su equivalente recursivo es:

```
int sumaDigitosRecur(int n)
{
   int res;

   if (n < 10)
        res = n;
   else
        res = n%10 + sumaDigitosRecur(n/10);

   return(res);
}
```

Ejercicio Propuesto 1.

a) Realiza varias trazas de la siguiente función recursiva.

```
int recur(int n)
{
   int res;

   if (n < 10)
         res = n;
   else
         res = (n%10)*(n%10) + recur(n/10);

   return(res);
}
```

b) Explica en lenguaje natural qué hace la función.

Ejercicio Propuesto 2. **Implementa un programa en C que incluya el módulo recursivo** `void imprimeNumeros(int inicio, int fin, char orden)` **que imprima en pantalla los números desde** `inicio` **hasta** `fin` **en orden ascendente o descendente según indique el carácter** `orden` **(**'c' **o** 'd' **según sea creciente o decreciente).**

Ejemplo:

`imprimeNumeros(2,7,'c')` → `2 3 4 5 6 7`

`imprimeNumeros(5,2,'d')` → `5 4 3 2`

`imprimeNumeros(5,7,'d')` → no imprime nada

Ejercicio Propuesto 3. **Implementa un programa en C que incluya el módulo recursivo:** `double potencia (double a, int b)` **que calcule la potencia a^b. Evidentemente, no puedes utilizar la función** `pow`**, debes utilizar el operador multiplicador *. Previamente debes escribir su definición matemática.**

Ejemplo:

`Potencia(2,3)` → 8

Ejercicio Propuesto 4. **Implementa la función recursiva** `int sumaMult5(int n)` **que devuelva la suma de los múltiplos de 5 que hay desde 1 hasta n. Por ejemplo: si n es el número 22 devolverá 50 (ya que los múltiplos de 5 que hay hasta 22 son 5, 10, 15 y 20 y su suma es 50).**

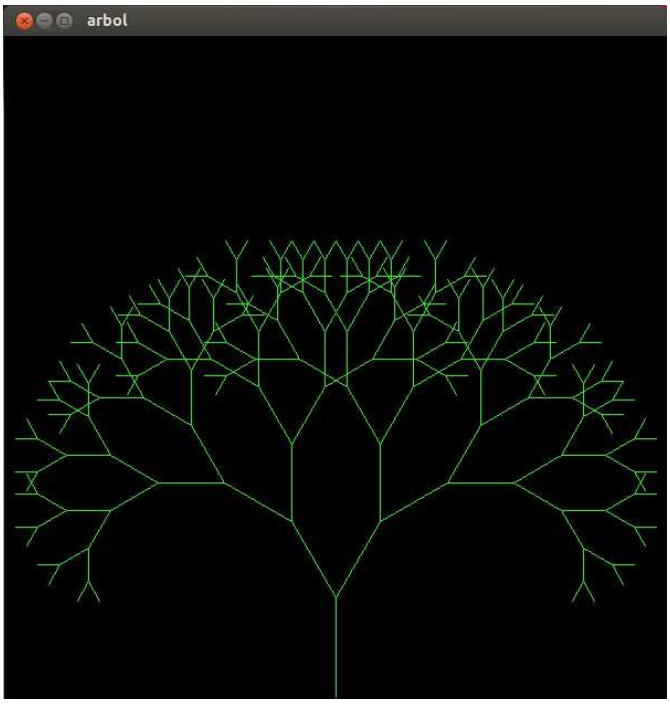
Ejercicio Propuesto 5. **Escribir un módulo recursivo** `void parYmayorN(int n, int elem)` **que imprima por pantalla los números desde 1 hasta** `n` **que son pares y mayores que** `elem`**. Por ejemplo, si** `n` **es 10 y** `elem` **es 4, imprimirá por pantalla 6, 8, 10.**

Ejercicio Propuesto 6. **Implementa el equivalente recursivo de la siguiente función y explica en lenguaje natural qué hace:**

```
int funcion(int a, int b)
{
  int res = 0;

  while(b > 0)
  {
    res += a;
    b--;
  }
  return res;
}
```

Ejercicio Propuesto 7. **Haciendo uso de los gráficos de tortuga, haz un programa que dibuje recursivamente el "árbol" que aparece en la siguiente figura:**



Fíjate en que cada nivel del árbol está formado por una rama que se divide en dos, y lo que tenemos en cada división es una especie de versión más pequeña del árbol completo. Por lo tanto, tendrás que:

- **Dibujar la rama (una línea recta de determinada longitud)**
- **Girar la tortuga sobre sí misma un determinado número de grados a la izquierda, por ejemplo 30 para disponerte a dibujar el "subárbol" de la izquierda.**
- **Dibujar un árbol un poco más pequeño que el tamaño actual**
- **Girar la tortuga a la derecha para disponerte a dibujar el "subárbol" de la derecha (serán -60, ya que tienes que deshacer los 30 que giraste y girar otros 30 más)**
- **Dibujar un árbol un poco más pequeño que el tamaño actual**
- **Volver a girar a la izquierda 30 para quedarte mirando a la horizontal**
- **Volver hacia atrás al punto de partida (en línea recta pero una cantidad negativa, usa la orden "move" en lugar de "draw" ya que no necesitas dibujar)**

Ten en cuenta que debes posicionar la tortuga en la parte central inferior de la ventana y empezar a dibujar el "tronco". Dado que la tortuga empieza mirando a la derecha, tendrás que girarla 90 grados a la izquierda antes de empezar a dibujar.
Prueba a variar la longitud de la rama o el número de grados, o a hacer árboles asimétricos en los que el número de grados que se gira para el subárbol izquierdo sea distinto al derecho. Mejor aún, modifica el programa para que el usuario pueda introducir los parámetros del árbol (longitud, ángulo izquierdo y ángulo derecho) por teclado.

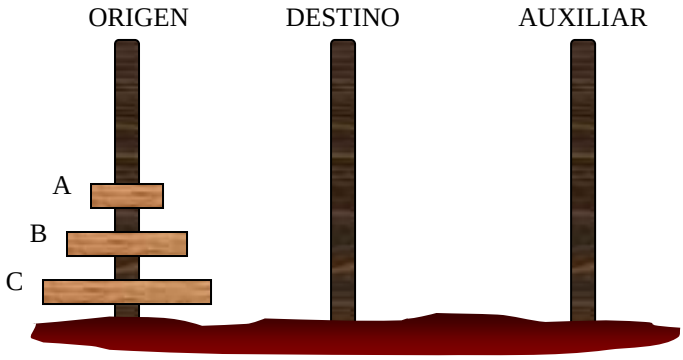
ERRORES DE PROGRAMACIÓN COMUNES

- **Definir una función recursiva que no tiene caso base.** Toda función recursiva debe tener una rama de salida que permita poner fin a la recursividad. En caso contrario, el programa no acabará nunca.
- **No modificar ningún parámetro en la llamada recursiva.** Al menos uno de los parámetros tiene que ir cambiando en las sucesivas llamadas recursivas de forma que se tienda a que se cumpla el caso base

Curiosidad: El problema de las torres de Hanoi

Durante algunos siglos los habitantes de un monasterio en Hanoi no tenían apenas relación con el mundo exterior y ello les llevó a idear algún modo de entretenimiento. Construyeron tres largos postes que colocaron en el jardín y otros se encargaron de construir distintos discos de madera de varios tamaños. El entretenimiento les llevaba varias horas al día y bastantes meses, pues no conocían la estrategia adecuada para finalizar rápidamente.

Hemos pensado que podrías ayudarles con un sencillo algoritmo recursivo que resolviese su juego. ¿Probamos? El juego de las torres de Hanoi consiste en tener unos discos colocados del siguiente modo:



Dichos discos están perforados y se trata de pasarlos al poste llamado DESTINO, pero de forma que en ningún momento quede un disco colocado sobre otro de tamaño inferior, para lo que se puede emplear temporalmente el poste AUXILIAR.

```
#include <iostream>

using namespace std;

void transferir(int,char,char,char);

int main(void) {
   int n;
   cout << "Bienvenido a las torres de Hanoi"<<endl;
   cout << "Cuántos discos?";
   cin >> n;
   transferir(n,'I','D','C');
}


// La función devuelve un número entero
void transferir(int discos, char desde, char hacia, char temp)
{
   if (discos>0)
   {
        transferir(discos-1,desde,temp,hacia);
        cout << "Mover disco '"<<discos<<"' desde "<<desde<<" hasta
"<<hacia<<endl;
        transferir(discos-1,temp,hacia,desde);
   }
}
```